Finalized Security Assessment

LOCKON Finance

Oct 25th, 2024

This smart contract audit report was created by Bunzz Audit. It utilized a database of over 1000 contract vulnerability patterns, comparing the project's contract against this database with AI to conduct a comprehensive diagnosis of vulnerabilities

BUNZZ AUDIT

Table of Contents

Summary

The audited smart contract is simple and high-quality, and no critical or high severity issues were found. A low severity issues were identified and discussed with the LOCKON Finance team.

Source URLs: https://github.com/lockon-finance/lock-contracts/blob/f1cd3177adc086f105ec6d480c2b14c734f969f3/contracts/LockonReferral.sol

Coverage

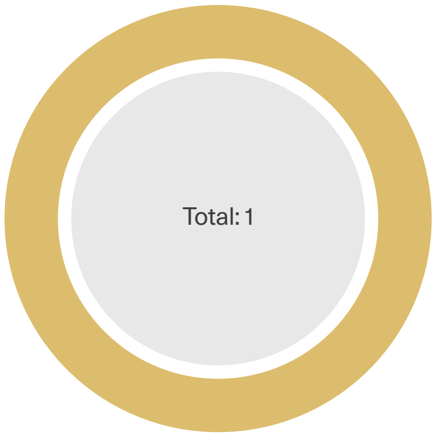
In this audit, we checked a total of 72 types of vulnerability patterns.

🟥	Found	1 (1.39%)
🟩	Not Found	71 (98.61%)

Title	Found
Front Running	-
Back Running	-
Sandwiching	-
Transaction order dependency	-
Fake tokens	-
Fake contracts	-
On-chain oracle manipulation	-
Governance attack	-
Token standard incompatibility	-
Flash liquidity borrow, purchase, mint, or deposit	-
Unsafe call to phantom function	-
One DeFi protocol dependency	-
Unfair slippage protection	-
Unfair liquidity providing	-
Unsafe or infinite token approval	-
Delegatecall injection	-
Unhandled or mishandled exception	-
Locked or frozen tokens	-
Absence of code logic or sanity check	-
Casting	-
Unbounded operation, including gas limit and call-stack depth	-
Arithmetic mistakes	-
Inconsistent access control	-
Visibility errors, including unrestricted action	-
Direct call to untrusted contract	-
Function Default Visibility	-
Integer Overflow and Underflow	-
Outdated Compiler Version	-
Floating Pragma	-

Unchecked Call Return Value	-
Unprotected Ether Withdrawal	-
Wrong Comparison Operator	-
UINT256 could be more gas efficient than smaller types	-
Magic numbers should be replaced with constants	-
Unprotected SELFDESTRUCT Instruction	-
Reentrancy	-
State Variable Default Visibility	-
Uninitialized Storage Pointer	-
Assert Violation	-
Use of Deprecated Solidity Functions	-
Delegatecall to Untrusted Callee	-
DoS with Failed Call	-
Transaction Order Dependence	-
Authorization through tx.origin	-
Block values as a proxy for time	-
Signature Malleability	-
Incorrect Constructor Name	-
Shadowing State Variables	-
Weak Sources of Randomness from Chain Attributes	-
Missing Protection against Signature Replay Attacks	-
Lack of Proper Signature Verification	-
Requirement Violation	-
Write to Arbitrary Storage Location	-
Incorrect Inheritance Order	-
Insufficient Gas Griefing	-
Arbitrary Jump with Function Type Variable	-
DoS With Block Gas Limit	-
Typographical Error	-
Right-To-Left-Override control character (U+202E)	-
Presence of unused variables	-
Unexpected Ether balance	-
Hash Collisions With Multiple Variable Length Arguments	-
Message call with hardcoded gas amount	-
Code With No Effects	-
Unencrypted Private Data On-Chain	-
Constant variables should be marked as private	-
Reading array length in for-loops	-
Checked arithmetic for for-loops	-
++i costs less gas than i++	-
IERC20.transfer does not support all ERC20 tokens	-
Unrestricted minting	-
Trust Issue Of Admin Roles	ISSUE-1

Findings



🟥	Critical	0 (0.00%)
🟥	High	0 (0.00%)
🟧	Medium	0 (0.00%)
🟨	Low	1 (100.00%)
🟦	Note	0 (0.00%)

ID	Title	Severity	Status
ISSUE-1	Centralized Owner Control Enables Arbitrary Token Withdrawal	● Low	Acknowledged

Details

ISSUE-1: Centralized Owner Control Enables Arbitrary Token Withdrawal

Severity

Low

Location

contracts/LockonReferral.sol:LockonReferral:deallocateRewardToken

Description

The ` deallocateRewardToken` function permits the owner to withdraw tokens from the contract. If the owner withdraw too much amount of token, users fail to claim pending reward.

How to fix

Implement a mechanism to ensure the owner cannot withdraw tokens arbitrarily. Consider keeping the pending reward by creating a on-chain signing mechanism.

Code

```
function deallocateRewardToken(
    address _tokenAddress,
    uint256 _amount
) external onlyOwner nonReentrant {
    require(
        _tokenAddress != address(0),
        "LOCKON Referral: Zero address not allowed"
    );
    require(
        isValidRewardToken(_tokenAddress),
        "LOCKON Referral: _tokenAddress not supported"
    );
    require(
        _amount > 0,
        "LOCKON Referral: _amount must be greater than Zero"
    );
    SafeERC20.safeTransfer(IERC20(_tokenAddress), msg.sender, _amount);
    emit RewardTokenDeallocated(msg.sender, _tokenAddress, _amount);
}
```

Code Suggestion

```
function deallocateRewardToken(
    address _tokenAddress,
    uint256 _amount
) external onlyOwner nonReentrant {
    require(
        _tokenAddress != address(0),
        "LOCKON Referral: Zero address not allowed"
    );
    require(
        isValidRewardToken(_tokenAddress),
        "LOCKON Referral: _tokenAddress not supported"
    );
    require(
        _amount > 0,
        "LOCKON Referral: _amount must be greater than Zero"
    );
     require(
        IERC20(_tokenAddress).balanceOf(address(this)) - _amount >=
pendingReward[_tokenAddress],
        "LOCKON Referral: Pending reward needs to be kept"
    );
    SafeERC20.safeTransfer(IERC20(_tokenAddress), msg.sender, _amount);
    emit RewardTokenDeallocated(msg.sender, _tokenAddress, _amount);
}


function signForReward(..., address _tokenAddress, uint256 _amount) public
onlyOwner returns (bytes){
    // sign for reward

    // update pendingReward
    pendingReward[_tokenAddress] += _amount;
    return signature;
}
```

Status

Acknowledged

Comment

The project team is aware of this issue and decided not to revise the code as a design choice. The impact is limited because of the following reasons.
- The owner is a trusted party.
- This contract is not what users deposit funds into.
- This is not an issue that an external party can exploit.

Disclaimer

We conducted our review of the smart contract codes solely based on the materials and documentation provided by the project under audit (the "Project").

Our audit employed a fine-tuned Artificial Intelligence (AI) system, which incorporates (i) a database of known vulnerability patterns that we have collected up until the date of our review for this audit, and (ii) results from a selection of existing contract analysis tools available as of the date of our review for this audit. While we endeavor to ensure the highest possible quality and accuracy of the results produced by our fine-tuned AI, we must clarify that the results are based on the state of the AI technology and understanding of smart contracts as of the date mentioned, and consequently absolute completeness and infallibility of the AI-generated results cannot be guaranteed.

In addition to the aforementioned AI-driven analysis, we may conduct manual audits. These are grounded in the knowledge and expertise we have accumulated up to the date of our review for this audit. The purpose of these manual audits is to identify and assess vulnerabilities specific to the project under audit.

Blockchain and smart contract technologies continue to evolve and may be susceptible to unforeseen risks and flaws. Consequently, while it is possible to minimize smart contract security risks, their complete elimination is inherently unattainable. Therefore, our audit does not claim to provide an exhaustive or all-encompassing review of all potential vulnerabilities.

Lastly, it is important to clarify that the scope of our audit is strictly limited to the analysis of smart contracts. Our audit does not extend to other layers or components, including but not limited to hardware, operating systems, programming languages, compilers, protocols, platforms, virtual machines, and imported libraries.

DISCLAIMER: You agree to the terms set forth in this disclaimer by reading this report or any part of it. Bunzz Pte. Ltd. and its affiliates (including shareholders, subsidiaries, employees, directors, and other representatives if any) (“Bunzz”) provide this report for information purposes only. No party, including third parties, has the right to rely on this report or its contents. Bunzz assumes no responsibility or duty of care to any person and makes no warranty or representation about the accuracy or completeness of this report to any person. Bunzz provides this report "as-is," without any conditions, warranties, or other terms of any kind, and hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, fitness for purpose, merchantability, or non-infringement). Except to the extent that it is prohibited by law, Bunzz excludes all liability and responsibility, and you or any other person will not have any claim against Bunzz for any amount or kind of loss or damage that may result to you or any other person (including, without limitation, any direct, indirect, special, punitive, consequential, or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, whether in tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report, its use, the inability to use it, the results of its use, and any reliance on this report. For the avoidance of doubt, no one should consider or rely upon this report, its content, access, or usage as any form of financial, investment, tax, legal, regulatory, or other advice.